Objektno orijentirano programiranje - praktikum u .NET okolini

Predavanje 08

Teme

- Predlošci
- *Binding*
- MVVM

Predlošci kontrola

- Većina Windows Forms kontrola su omotači oko Win32 API-ja koji je nepromjenjiv
- WPF kontrole su potpuno napisane u .NET-u
  - Možemo im mijenjati izgled po želji
- **Predlošci kontrola** (engl. *control templates*) omogućuju izmjenu podrazumijevanog izgleda WPF kontrole
  - Svaka WPF kontrola ima svojstvo Template koje je tipa ControlTemplate
  - Postavljanjem svojstva na novu vrijednost mijenjamo izgled kontrole
    - Ponašanje ostaje nepromijenjeno!

Predlošci kontrola

- Svaka WPF kontrola ima svoj podrazumijevani predložak koji definira od kojih vizualnih elemenata se kontrola sastoji
- Da bismo elementu promijenili predložak, obično postupamo na sljedeći način:
  1. Definiramo predložak kao resurs tipa `ControlTemplate`
  2. Na željene elemente primjenjujemo predložak

- Moguće je primijeniti predložak putem stila tako da definiramo stil koji postavlja svojstvo Template na novi predložak

Elementi predloška

- Definicija predloška obično se sastoji od:
  - Ciljnog elementa za kojeg je definiran (svojstvo `TargetType`)
  - Elemenata od kojih se sastoji (vizualno stablo)
  - Okidača koji definiraju dinamiku

Elementi predloška

- Elemente vizualnog stabla često imenujemo kako bismo im mogli pristupiti iz okidača
  - Elementu dajemo naziv pomoću atributa **`x:Name`**
  - Elementu pristupamo pomoću atributa **`TargetName`**
- Kontrola za koju izrađujemo predložak ima definirano niz svojstava i njihovih vrijednosti
- Želimo li pojedina svojstva unutar predloška postaviti na kontroli na koju se primjenjuje predložak
  - Koristimo XAML proširivač **`TemplateBinding`** i njegovo svojstvo **`Property`** kako bismo uzeli vrijednost s kontrole

```
<ContentPresenter
    HorizontalAlignment="{TemplateBinding Property=HorizontalAlignment}"/>
```

Povezivanje

- Povezivanje (engl. *binding*) je odnos koji kaže WPF-u da uzme podatak iz izvornog (engl. *source*) objekta i postavi ga kao vrijednost ovisnog svojstva ciljnog (engl. *target*) objekta
  - Izvorni objekt može biti bilo što: WPF element, DataRow, instanca klase, …
  - Ciljno svojstvo je uvijek ovisno svojstvo
- Proći ćemo dva tipa povezivanja:
  - Izvorni objekt je neki WPF element
    - Svojstvo iz kojeg uzimamo podatke je onda ovisno svojstvo
  - Izvorni objekt je bilo koji .NET objekt

Definiranje povezivanja kroz XAML

- Povezivanje najčešće definiramo kroz XAML
- Izraz povezivanja (engl. *binding expression*) definira odakle i kako uzimamo podatak
  - Definiramo ga XAML proširivačem **`Binding`**
  - Izvorni element referenciramo svojstvom **`ElementName`**
  - Izvorno svojstvo referenciramo svojstvom **`Path`**
  - Smjer povezivanja definiramo svojstvom **`Mode`**:
    - `OneWay, OneWayToSource, **TwoWay**, OneTime`
- U slučaju problema s definiranim povezivanjem, WPF neće baciti iznimku
  - Detalje o pogreškama možemo pronaći u *debug* prozoru

Definiranje povezivanja kroz kôd

- Klasa `Binding` predstavlja objekt povezivanja:
  - Izvorni element referenciramo svojstvom **`Source`**
  - Izvorno svojstvo referenciramo svojstvom **`Path`** i postavljamo ga na instancu tipa **`PropertyPath`**
  - Smjer povezivanja definiramo svojstvom **`Mode`** i postavljamo ga na vrijednost enumeracije **`BindingMode`**
- Kad smo pripremili objekt, na ciljnom elementu pozivamo metodu **`SetBinding()`**

Povezivanje s elementima

- Najjednostavniji oblik povezivanja je povezivanje s WPF elementima
- Primjerice, povežimo `TextBox` sa `Sliderom`:

```
<Slider x:Name="slider"
        Minimum="1" Maximum="100"
        Value="50" TickFrequency="10" Margin="5"
        TickPlacement="BottomRight" />
<TextBox Margin="5"
         Text="{Binding ElementName=slider, Path=Value}" />
```

- Promjenom `Slidera`, ažurira se sadržaj `TextBoxa`
- Ali, i promjenom vrijednosti u `TextBoxu`, ažurira se `Slider`
  - Podrazumijevani način je `TwoWay`
  - Tek nakon gubitka fokusa (svojstvo `UpdateSourceTrigger`)

Primjer povezivanja s elementima

- Definirajmo okomiti i horizontalni `Slider` koji omogućuje pomicanje kruga
- Krug treba moći pomicati po cijelom roditeljskom kontejneru

```
<DockPanel LastChildFill="True">
    <Slider x:Name="h"
            DockPanel.Dock="Top"
            Minimum="0"
            Maximum="{Binding ElementName=canvas, Path=Width}"
            Value="0" />
    <Slider x:Name="v"
            DockPanel.Dock="Left"
            Minimum="0"
            Maximum="{Binding ElementName=canvas, Path=Height}"
            Value="0"
            Orientation="Vertical"/>

    <Canvas x:Name="canvas" Width="400" Height="200"
            HorizontalAlignment="Left"
            VerticalAlignment="Top">
        <Ellipse x:Name="ell"
                 Canvas.Left="{Binding ElementName=h, Path=Value}"
                 Canvas.Top="{Binding ElementName=v, Path=Value}"
                 Fill="Purple"
                 Width="100"
                 Height="100"/>
    </Canvas>
</DockPanel>
```

Povezivanje s podacima svojstava .NET objekata

- Kako bi se promjene vrijednosti svojstava .NET objekta manifestirale na korisničkom sučelju (UI), treba za povezivanje koristi **`MVVM`** (engl. *model-view-viewmodel*) obrazac:
  - Podatak se enkapsulira u pogledni model (**`VM`**) koji implementira **`INotifyPropertyChanged`** sučelje
  - **`VM`** se veže na roditelja pomoću svojstva **`DataContext`**
  - Kontrole se referenciraju na svojstva poglednog modela povezivanjem
    - Koristi se samo svojstvo **`Path`** jer je izvor podataka **`DataContext`**

MVVM – Model-View-ViewModel

- **Presentational pattern**
  - Popravlja čvrstu povezanost Modela i Viewa u MVC patternu
    - Podaci za prezentaciju predstavljeni su **ViewModel** entitetom koji enkapsulira sve podatke potrebne prezentaciji
    - Može povezivati i više entiteta
- **Observable pattern**
  - WPF omogućuje implementaciju INotifyPropertyChanged sučelja koje omogućuje povezanom elementu primanje notifikacija promjena i ažurira se u skladu s istima